TP2 | Over the Top Content Delivery

José Pedro Fonte[1 [a91775]], Bernard Ambrosio Georges[2 [pg53698]], and Vasco Manuel Oliveira[3 [pg54269]]

[1] Universidade do Minho
[2] Mestrado em Engenharia Informática
[3] Engenharia de Serviços em Rede

Abstract. O projeto atual aborda a criação de um serviço Over the Top (OTT) para entrega eficiente de conteúdo multimédia em tempo real. O foco está na utilização de uma rede overlay aplicacional para distribuir áudio, vídeo e texto de um servidor para múltiplos clientes. Como tal, foi utilizado o emulador CORE como plataforma de testes e uma variedade de topologias experimentais.

Keywords: OTT (Over the Top) · Rede *Overlay* Aplicacional · Conteúdo Multimédia.

1 Introdução

O Presente relatório tem como objetivo demonstrar e explicar o percurso feito pelo grupo ao implementar a rede OTT. Uma rede OTT é uma rede que permite a distribuição de conteúdo de multimédia diretamente aos clientes por meio da internet, sem necessidade de uma infraestrutura de distribuição tradicional.

Ao longo deste relatório, será discutido a arquitetura da solução proposta, a especificação dos protocolos utilizados, os detalhes da implementação, limitações encontradas durante o processo, os testes realizados e os resultados obtidos, além de conclusões relevantes e possíveis direções para trabalho futuro.

2 Arquitetura da solução

Nesta secção, iremos apresentar a arquitetura proposta para o serviço de entrega de conteúdo multimédia OTT, falando das diferentes componentes que a constituem e as suas interações, e mostrando a topologia overlay proposta. A Figura 1 ilustra a topologia da rede overlay aplicacional desenvolvida para o projeto.

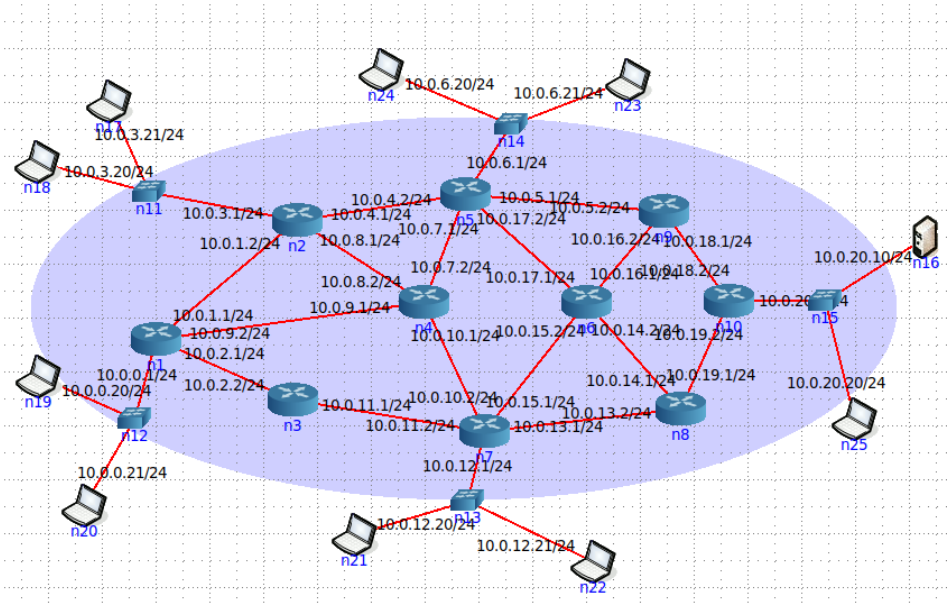

Fig. 1. Topologia de Rede Principal

A nossa arquitetura é composta por diferentes elementos interligados, cada um desempenhando um papel crucial na transmissão eficiente de conteúdo para os clientes:

1. **Rendezvous Point (RP):** Este ponto central na rede é responsável por receber o conteúdo a ser distribuído em unicast, propagando-se para os nós intermediários. É o núcleo da árvore de distribuição, otimizando a entrega dos conteúdos de maneira eficiente, com menor atraso. Na nossa topologia este será representado na rede *underlay* pelo nó "n7", ilustrado na Figura 1.
2. **Nós Overlay:** São os nós intermediários que compõem a rede overlay aplicacional. Estes nós são responsáveis pela distribuição eficiente do conteúdo, formando uma árvore de distribuição compartilhada para garantir uma transmissão eficiente dos dados. Na nossa topologia estes são representados na rede *underlay* pelos nós ""n1", "n2", "n9" e "n10", ilustrados na Figura 1.
3. **Clientes:** Representam os destinatários finais do conteúdo multimédia. Estes clientes enviam solicitações para os servidores da rede overlay para receberem a stream desejada. Na nossa topologia estes são representados na rede *underlay* pelos nós ""n17", "n18", "n22" e "n23", ilustrados na Figura 1.
4. **Servidores de Conteúdo:** São os pontos de origem dos dados multimédia. Responsáveis por armazenar e fornecer o conteúdo aos clientes interessados. Estão ligados diretamente com o Rendezvous Point para poderem fornecer conteúdos de forma mais eficiente. Na nossa topologia estes são representados na rede *underlay* pelos nós "n16" e "n19", ilustrados na Figura 1.

3 Especificação do(s) protocolo(s)

Para a implementação da rede, optou-se por uma estratégia que divide as mensagens em duas categorias principais: mensagens de controle e mensagens de streaming. Essa abordagem visa otimizar o fluxo da rede, garantindo uma comunicação eficiente entre os diferentes componentes do sistema.

No que diz respeito às mensagens de controle, preferiu-se o uso do modelo de unicast, onde cada mensagem é enviada individualmente para o seu destinatário específico. Para este modelo favoreceu-se o uso do protocolo TCP, pois este oferece confiabilidade na entrega das mensagens, garantindo que cada comando de controlo seja recebido corretamente pelo destinatário, mesmo em ambientes de rede sujeitos a perdas de dados.

Por outro lado, para as mensagens de streaming, adotou-se uma outra abordagem. Neste caso, de modo a priorizar uma transmissão rápida e contínua do conteúdo foi utilizado o protocolo UDP. O UDP, por ser um protocolo de transporte sem conexão, oferece uma transmissão mais ágil e eficiente dos dados, sendo mais suscetível a eventuais perdas de pacotes.

3.1 Formato das mensagens protocolares

Para implementar um formato padronizado nas mensagens, foi empregada a função 'messageBuilder' e 'messageDecoder'. Estas funções codificam/descodificam o tipo, o subtipo e os dados desejados a serem incorporados/retirados da mensagem. A mensagem reserva os primeiros dois bytes para o tipo e subtipo, enquanto o restante é dedicado à inclusão das informações essenciais a serem transmitidas.

3.2 Interações

Na rede implementada, foram incorporados tipos distintos de interações possíveis:

Em conexões TCP

1. **Mensagens de Controle:** São mensagens usadas para o conhecimento da rede e integração na mesma. Este tipo pode ser dos seguintes subtipos
   - bootstraper -> mensagens para integração na rede feitas ao RP com retorno de seus nodos vizinhos.
   - connect -> mensagens de conexão. Enviada aos nodos de modo a informar a sua ligação a rede.
   - rp path -> usada para encontra o melhor caminho em direção ao RP.
2. **Mensagens de Conteúdo:**
   - vídeos catalog -> usada Pelo RP de modo a catalogar os vídeos disponíveis em cada Servidor de conteúdo.
   - content sub -> usada para a inicialização de um stream.
   - content term -> usada para sinalizar a finalização da stream.

Em conexões UDP

1. **Transmissão de Pacotes de Conteúdo**

4 Implementação

4.1 Rendezvous Point (RP)

O nó RP serve tanto como um servidor normal como um bootstrapper, para isto, o RP tem um ficheiro JSON onde guarda a topologia overlay. Na sua inicialização, o nó lê o ficheiro e guarda o seu conteúdo na struct NetworkGraph. A partir desta estrutura é construído uma lista de nodos vizinhos, struct NeighbourNodes, que utiliza o struct DirectConnection como o tipo dos seus elementos. O DirectConnection tem como objetivo definir os nós da topologia contendo por isso o IP do nó, as suas métricas e uma Label usada como um identificador.

Assim que o Rendezvous Point (RP) guarda a lista dos seus nós vizinhos, é criada uma goroutine para o cálculo de métricas e a disponibilidade de vídeos. Essa rotina tem como objetivo manter as informações atualizadas, reavaliando os pedidos a cada intervalo de tempo aleatório, onde o valor de X é um número variando entre 10 e 21.

Posteriormente, o RP estabelece a sua conexão TCP, e a cada nova conexão, uma nova rotina, que representada pela função handleTCPrequests, é encarregada de lidar com as mensagens lidas da conexão TCP. Essa rotina é responsável por responder ás mensagens de controlo e de conteúdo apresentadas na conexão. A do tipo inicial é subdivida em três subtipos: **bootstrapper** que apresenta ao remetente a lista dos seus nós vizinhos, **connect** que é o pedido vindo de um nó vizinho de se conectar com o mesmo e, finalmente, **RP path**, este simboliza o fim do protocolo apresentado no Nó overlay, visto que o retorno deve ser o score da sua conexão, permitindo, assim, que os nós avaliem o melhor caminho ao RP. Já o segundo tipo traduz-se no pedido de streaming, em que se difere do apresentado nó de overlay no RP gateway, que neste caso será o servidor de conteúdo do vídeo desejado.

4.2 Nós Overlay

Inicialmente, é essencial enviar uma solicitação ao bootstrapper. Essa ação permite conhecer os IPs para a criação de conexões com cada vizinho, que por sua vez, permite a integração do novo nó na topologia overlay.

Após a integração, o nó realiza a busca pelo caminho mais eficiente até ao Rendezvous Point (RP). Nesse processo, o nó envia mensagens de controle do tipo "RP path" a todos os seus vizinhos, aos quais, por sua vez, volta a passar essas mensagens recursivamente até atingirem o RP. Quando o RP responde com a avaliação da ligação, o percurso é revertido. Nesse sentido, é adicionada a avaliação do nó anterior ao valor da ligação entre eles. Ao final desse processo, o nó seleciona o nodo com a melhor avaliação para ser designado como o gateway para o RP.

Por fim, o nó estabelece uma conexão com os clientes. Se estiver atualmente transmitindo o vídeo solicitado, o cliente é adicionado à lista de destinatários do vídeo. Caso não tenha streams ativas, o nó encaminha a solicitação ao seu RP gateway. Este processo repete-se até atingir um nó com a stream desejada e

como último recurso chega ao RP, em que este requisita ao servidor de conteúdo o video. Como resposta, o servidor inicia uma stream unicast do video ao RP que, em contrapartida, manda como resposta ao nó inicial uma mensagem de reconhecimento e inicia a stream multicast do vídeo.

4.3 Servidor de Conteúdo

Como anteriormente explicado, o servido de conteúdo tem como seu objetivo o streaming unicast dos seus vídeos armazenados para o RP. Como tal, este tem apenas conexão com o Rendezvous Point, que se caracteriza apenas pelos seus pedidos de transmissão dos vídeos armazenados.

4.4 Cliente

À semelhança dos outros componentes, o cliente inicia estabelecendo uma conexão com o nó overlay desejado. Após a concretização dessa ligação, o cliente solicita o streaming do vídeo desejado, aguardando como retorno a mensagem de reconhecimento do nó. Ao receber esta mensagem, o cliente se conecta a um socket usando o seu IP e porta, tornando-se um ouvinte das mensagens UDP nessa porta.

Imediatamente, o cliente inicia a leitura das mensagens do socket, inserindo-as no ffplay. Além disso, o cliente define o standard output como a saída do ffplay, permitindo assim que o vídeo seja exibido na tela do cliente. Finalmente, caso o cliente deseje encerrar o streaming, ele envia ao seu nó um pedido de turn, sinalizando o término das transmissões para este cliente.

4.5 Bibliotecas de funções

Um dos principais requisitos apresentados é a apresentação do serviço de entrega do conteúdo multimédia em tempo real, de modo a facilitar a sua concretização, optou-se pelo uso da linguagem Go, que, devido à sua eficiência, desempenho e suporte robusto de concorrência, permite-nos atender aos requisitos de tempo real de uma forma mais simples.

Para além de suas funções nativas foi necessário o uso da varias bibliotecas. Uma das mais importantes foi a biblioteca **net**, nativa de go, esta biblioteca permitiu a criação, escrita e leitura de conexões UDP e TCP. Para além desta também foi usado o **fmt** para o debugging e o **os** para os argumentos de inicialização de cada nó. Também foi usadas algumas bibliotecas para o casting de variáveis e outras funções mínimas.

4.6 Parâmetros

Para inicializar cada componente da rede overlay, deve-se seguir os seguintes formatos:

- **Cliente:**

  - IP do nó do overlay ao qual deseja se conectar.
  - Id do nó.
  - exemplo: ./client 10.0.1.1 n17
- **ServerRP (Rendevouz Point):**
  - Id do nó.
  - exemplo: ./serverRP n7
- **Server (nó):**
  - IP do RP.
  - Id do nó.
  - exemplo: ./serverRP 10.0.10.2 n1
- **Servidor de Conteúdo:**
  - IP do RP.
  - Id do nó.
  - exemplo: ./serverCont 10.0.10.2 n16

Na execução de cada elemento também é importante respeitar uma ordem, sendo esta, RP -> Nós -> Servidor Conteúdo -> Clientes.

5 Limitações da Solução

No desenvolver deste projeto o grupo encontrou algumas limitações. Primeiramente devido a pouca familiaridade com a linguagem GO foi notada uma curva de aprendizagem e uma maior dificuldade na implementação eficaz do código desejado, juntamente com a isso a sua falta de uma documentação detalhada, mais especificamente exemplos implementados, significou uma prolongação do tempo de aprendizagem necessária. Outra dificuldade encontrada foi a utilização do o emulador CORE como ferramenta para testar o nosso serviço, pois levou algum tempo a descobrir como configurar os diferentes nós com o código feito.

Estas dificuldades em conjugação com o pouco tempo de desenvolvimento resultaram numa solução com algumas limitações técnicas, que a equipa está certa que seriam resolvidas dado mais algum tempo:

- Não é possível escolher conteúdo dos servidores.
- O cliente ao sair e entrar na stream afeta a sua qualidade.

-

6 Testes e resultados

Nesta secção vamos mostrar que a construção da rede overlay está efetivamente a acontecer, que o servidor RP está realmente a calcular e atualizar as métricas em relação às condições de entrega de cada um dos servidores, e que o streaming de vídeo está a ser realizado devidamente.

6.1 Resultados da Construção da rede overlay

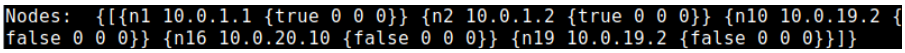

Fig. 2. Conexão dos vizinhos do nó RP

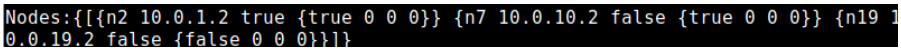

Fig. 3. Conexão dos vizinhos do Servidor n1

Nas Figuras 2 e 3 conseguimos ver os vizinhos dos nós n1 e n7 (que é o servidor RP), através das labels dos respetivos servidores vizinhos, e se as conexões com esses nós estão ativos, dando o valor *true* quando está ativo e o valor *false* caso contrário. Para este teste ligamos os servidores n1, n2, n9 e n7 para podermos mostrar que algumas conexões não estão ligadas.

6.2 Resultados das Métricas

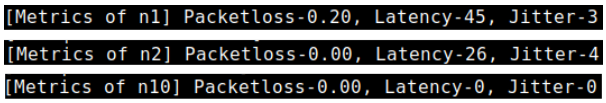

Fig. 4. Métricas dos servidores n1, n2 e n10

Na Figura 2 podemos ver as métricas "Packetloss", "Latency" e "Jitter" que foram calculadas sobre as condições de entrega, neste caso, nos servidores n1, n2 e n10. Para este teste adicionamos alguns efeitos de link, mais perto dos servidores n1 e n2, especificamente *Delay*, *Jitter* e *Loss*.

6.3 Resultados de Streaming

Fig. 5. Métricas dos servidores n1, n2 e n10

Na Figura 5 podemos ver os clientes n17, n18 e n24 a receberem a stream de vídeo ao mesmo tempo. Para este teste usamos o servidor de conteúdo n16, o servidor RP n7, o servidor intermediário n1 e os clientes n17, n18 e n24 que estão a fazer um pedido de stream para o servidor n1.

7 Conclusões e trabalho futuro

Em suma, a implementação da rede OTT representou um desafio significativo, mas também uma oportunidade para compreender a dinâmica complexa da entrega de conteúdo multimédia em tempo real.

Durante o processo, identificamos limitações que influenciaram o desenvolvimento, como a curva de aprendizagem da linguagem Go e a complexidade na configuração do emulador CORE. No entanto, essas dificuldades proporcionaram valiosos aprendizados sobre a implementação prática de redes OTT.

Os testes realizados forneceram resultados satisfatórios, validando a construção da rede overlay, a atualização das métricas de desempenho e a transmissão adequada do conteúdo entre os nós.

Como trabalho futuro, consideramos a exploração do aprimoramento do servidor de conteúdo para suportar uma maior variedade de vídeos para stream e a implementação de estratégias de recuperação de falhas.

Concluindo, este projeto ofereceu uma visão mais ampla da implementação de redes OTT e as suas nuances, fornecendo uma base sólida para eventuais desenvolvimentos nesta área.